Tarea 2: medidor ancho de banda UDP confiable usando Selective-Repeat
Redes
Plazo de entrega: 23 de octubre 2023

José M. Piquer

1. Descripción

Su misión, en esta tarea, es mejorar el cliente UDP de la Tarea 1, que permita medir ancho de banda total para recibir datos desde un servidor, aplicando un protocolo de corrección de errores basado en Selective-Repeat.

El servidor implementa un envío de paquetes con una ventana de tamaño fijo 50. El timeout para cada paquete será el que Uds le pasen al comienzo de la conexión.

Uds deben implementar un cliente con una ventana de recepción de tamaño 50. Si Uds usan el cliente de la Tarea1 con el nuevo servidor, verán que igual funciona. Pero si implementan bien el receptor con la ventana de 50, la performance que miden en Mbytes/s mejorará mucho.

Para tener un buen Selective-Repeat, mantienen un número de secuencia "esperado", pero también deben aceptar 50 paquetes más adelante de ese, y cualquiera que llegue lo ponen en su lugar en la ventana y envían el ACK específico para ese paquete. Una mejora importante de eficiencia es agregar un segundo tipo de ACK: un ack acumulativo, que implica que todos los paquetes anteriores a ese ya fueron recibidos bien. Un caracter 'A' es un ACK acumulativo (como en la tarea 1), en cambio una 'a' es un ACK parcial, sólo para ese paquete.

Entonces, cuando reciben un paquete de datos:

- si la secuencia no calza con la ventana de recepción: enviar 'A' para el último paquete antes del "esperado" y descartar el paquete de datos.
- si calza:
    - colocar en su lugar en la ventana

- si es el "esperado": correr la ventana todo lo posible (saltando todo paquete que tenga un ACK parcial) y enviar un ACK acumulativo.
- sino: enviar ACK parcial

Seguimos usando los mismos parámetros: el procentaje de pérdida y el valor del timeout en el servidor (que debe aproximar el máximo RTT: tiempo que toma ir y volver en la red entre cliente y servidor). El cliente que deben escribir recibe el tamaño de paquete a proponer, la cantidad de bytes a pedir al servidor, el valor del timeout (en ms) a pedir al servidor, la pérdida (en porcentaje), archivo a generar como salida, servidor, puerto:

```
./bwc-sr.py pack_sz nbytes timeout loss fileout host port
```

Hay un servidor corriendo en anakena.dcc.uchile.cl puerto 1818 UDP con este protocolo.

Pueden usar `sock.settimeout(val)` para programar el timeout del próximo `sock.recv()` que hagan. Si ponen `val=0` el socket se vuelve no-bloqueante y pueden preguntar si hay datos sin bloquearse. En este protocolo, lo usual es que siempre quieran bloquearse un rato esperando datos desde el servidor. Es él que tiene que implementar los timeouts inteligentes de retransmisión, Uds sólo deben generar los ACKs correspondientes. Si su lectura desde el servidor les da timeout, quiere decir que todo falló y deben terminar con un error, no tienen como corregir eso. Implementen un timeout para morir de 3 segundos.

Este protocolo es más complicado que el de la Tarea 1, pero no es necesario usar threads, ya que siempre están esperando datos, reaccionando a lo que reciben, y vuelta a esperar.

El cliente debe invocarse igual como en la tarea 1. Para comenzar a probar, les recomiendo usar 1.000 de tamaño de paquete, 100.000 bytes y 300 de timeout y 0 de pérdida:

```
% ./bwc-sr.py 1000 100000 300 0 out anakena.dcc.uchile.cl 1818
propuse paquete: 1000
recibo paquete: 1000
recibiendo 100000 nbytes
bytes recibidos=100000, time=1246.566123008728, bw=0.07650410987460893 MBytes/s
errores=51
```

2. Protocolo

1. Cliente: envía 'Cxxxxtttt'

2. Servidor: responde 'Cyyyy'
3. Cliente: envía 'Nnbytes'
4. Servidor: envía paquetes hasta completar 'nbytes' bytes en total. Siempre comienzan con 'D' y dos bytes que codifican el número de secuencia (entre 00 y 99) y son de tamaño máximo yyyy bytes más el header utilizar (o sea, resultan paquetes UDP de yyyy+3 como máximo tamaño): 'D00aaaaaaaaaaaaaaaaaaaaaaaa'
5. Cliente: envía ACK (acknowledgements) con el número de secuencia del último paquete recibido bien: 'A00'
6. Servidor: 'D01aaaaaaaaaaaaaaaaaaaaaaaa'
7. Cliente: 'A01'
8. ...
9. Servidor: 'D09aaaaaaaaaaaaaaaaaaaaaaaa'
10. Cliente: 'A09'
11. Servidor: 'D12aaaaaaaaaaaaaaaaaaaaaaaa'
12. Cliente: 'a12'
13. ...
14. Servidor: al completarse el envío, termina con un paquete 'E' con su número de secuencia en un paquete sólo y con eso marca el fin de la transmisión: 'E81'
15. Cliente envía último ACK: 'A81'

Si el EOF llega antes que la parte final del archivo, deben enviar un ACK parcial para él y seguir esperando a completar todos los datos.

3. Entregables

Básicamente entregar el archivo con el cliente que implementa el protocolo.

En un archivo aparte responder las preguntas siguientes (digamos, unos 5.000 caracteres máximo por pregunta):

1. Genere algunos experimentos con diversos tamaños de paquete, timeout y pérdidas y haga una recomendación de valores a utilizar para las distintas pérdidas. Grafique sus resultados.

2. Explique porqué el cliente de la tarea 1 igual funciona, a pesar que el servidor envía implementando selective-repeat y con una ventana de tamaño 50.

3. Calcule si el tamaño 50 de ventana está bien para la conexión que Ud tiene. Debería ser más? menos? por qué?

4. La ventana de recepción del cliente es igual a del enviador. ¿Serviría de algo que la ventana del cliente fuera más grande?